

# Scalability, timing, and system design issues for intrinsic evolvable hardware

**James Hereford**

Department of Physics and Engineering, Murray State University, Murray, KY, james.hereford@murraystate.edu

**David Gwaltney**

NASA – Marshall Space Flight Center, Huntsville, AL 35812, david.a.gwaltney@nasa.gov

**Keywords:** evolvable hardware, scalability, population sizing, timing analysis, design approaches

## Abstract

In this paper we address several issues pertinent to intrinsic evolvable hardware (EHW). The first issue is scalability; namely, how the design space scales as the programming string for the programmable device gets longer. We develop a model for population size and the number of generations as a function of the programming string length,  $L$ , and show that the number of circuit evaluations is an  $O(L^2)$  process. We compare our model to several successful intrinsic EHW experiments and discuss the many implications of our model. The second issue that we address is the timing of intrinsic EHW experiments. We show that the processing time is a small part of the overall time to derive or evolve a circuit and that major improvements in processor speed alone will have only a minimal impact on improving the scalability of intrinsic EHW. The third issue we consider is the system-level design of intrinsic EHW experiments. We review what other researchers have done to break the scalability barrier and contend that the type of reconfigurable platform and the evolutionary algorithm are tied together and impose limits on each other.

## 1.0 Introduction

Evolvable hardware (EHW) refers to hardware that can change its architecture and/or behavior dynamically, usually under the control of an evolutionary algorithm (EA) [1]. Evolvable hardware is usually implemented on a programmable or reconfigurable logic device, such as a field programmable gate array (FPGA) or a field programmable analog array (FPAA). The device is programmed by loading a string of bits. The architecture and thus the function of the programmable device are determined by the string of programming bits.

This paper addresses the issue of scalability of intrinsic evolvable hardware (EHW). We define intrinsic EHW as “on-line” evolution where the reconfigurable hardware platform is in the evolutionary loop. Intrinsic EHW thus takes into account any imperfections or peculiarities within the hardware device in calculating the final output. Several researchers have successfully derived electronic circuits using intrinsic evolvable techniques [2-11]. However, all of the known successful evolved circuits have been only small or, at best, medium-sized circuit; to our knowledge, no circuit with more than 100 functional elements has been evolved intrinsically.<sup>1</sup> But the programmable devices themselves can implement very complex circuits with many

---

<sup>1</sup> Thompson et al. [24] use 100 cells of a Xilinx FPGA to synthesize a tone discriminator but they are able to determine that only 32 out of the 100 cells are involved in generating the output.

functional elements. This raises the question of the scalability of intrinsic EHW; that is, how well the principles of intrinsic EHW will apply to circuits with 200, 1000, even 10,000 functional elements.

To address the issue of intrinsic EHW scalability, this paper will look at several topics. First, we will look at the design space and how big it gets as the programming string gets longer. We define the design space as the total space of allowable circuits given the hardware device(s). (The definition of design space includes both the number of programmable “configuration blocks” and the number of programmable interconnects among the blocks. To make this discussion general, a configuration block is defined as the smallest programmable component on a hardware device; it can be as simple as a transistor or as complicated as an amplifier or a logic block.) In general, the larger the design space then the more complex circuit that can be evolved. We assume that complex is a good thing since it means that the circuit will (potentially) be better than a simpler circuit performing the same function (e.g., faster response or sharper filter rolloff) or it will be capable of more functionality.

To evolve a complicated circuit requires a large design space and hence a long string of programming bits,  $L$ . (Roughly speaking, if no constraints are placed on the interconnects of the device, then the length of the programming string will grow in the order of  $O(nblocks)$  where  $nblocks$  is the number of configuration blocks in the design [1].) The size of the design space is  $2^L$  but an evolutionary algorithm (EA) does not necessarily need to do an exhaustive search. Instead, the EA usually only needs to evaluate a fraction of the total possibilities to derive a “good” circuit. (Note that there are some cases where a total search of the design space is required [12] but those cases are rare in intrinsic EHW.) To determine how many circuit evaluations are required to derive a good circuit, we develop equations for the population size,  $N$ , and the number of generations till convergence,  $ngen$ , given the length of the programming string. We then look at some implications that result from these equations and specifically address the question of whether the design space can become too big.

The second topic that will be addressed in this paper is the timing of intrinsic EHW applications. Intrinsic EHW is different from extrinsic or “off-line” EHW where the operation of the hardware device is simulated. This paper will look at the timing issues involved with intrinsic EHW and discuss “why does it take so long” to evolve a circuit intrinsically. We will look at two different intrinsic EHW setups and see how long each one takes to evaluate each member of the population and where the timing bottleneck occurs.

The third topic that we consider is what we call “system-level” design. In system-level design we consider the EA in tandem with the hardware platform in the design process rather than treating the two separately. We review what other researchers have done to break the “scalability barrier” and evolve larger, more complicated circuits. We contend that the evolutionary process and the type of reconfigurable platform are tied together and impose limits on each other. These two components of hardware evolution are a system to be considered together in the design of an intrinsic EHW process.

The outline for this paper is as follows: Section 1.1 discusses the unique aspects associated with intrinsic EHW as opposed to extrinsic EHW or standalone EAs. Section 2 develops theoretical

equations for the population size and number of generations till convergence based on the length of the programming string (chromosome),  $L$ . We compare these equations to results from successful intrinsic EHW experiments and see that there is good agreement between our model and the experimental results. From these equations, we can also estimate the number of circuit evaluations that are required to derive a “good” circuit intrinsically and look at implications of our model. Section 3 discusses the timing of intrinsic EHW experiments. Timing calculations are made for a relatively slow experimental configuration and a fast configuration. For both the slow and fast configurations, we show that increasing the processor speed does not significantly affect the overall timing. Instead, circuit evaluation time is the “long pole in the tent” and it must be reduced to see a significant reduction in intrinsic EHW timing. Section 4 looks at some of the approaches used to evolve larger circuits. In general, the current state of the practice is to limit the search space in some way to evolve/derive larger circuits. This section also discusses the issue of designing the evolutionary algorithm in conjunction with the reconfigurable platform in a system level design approach. Section 5 gives the main conclusions of the paper.

### 1.1 Issues germane to intrinsic evolvable hardware

There are several issues unique to intrinsic evolvable hardware; that is, these issues separate intrinsic EHW from extrinsic EHW or running EAs standalone.

The first issue is that not all the programming bits have equal validity or equal importance in determining the output. Specifically, the routing bits, which determine which functional elements/configuration blocks are used within the hardware platform must be set before other adjustments can be made. The output of the device is essentially random until the routing bits channel the input signal(s) from the input port to the output port. After the routing bits have been set (either *a priori* by the user or by the EA), then the other bits can be evolved to give the proper gain, wave shaping, or logic equation that is required.

The second issue separating intrinsic EHW is that the timing aspects associated with the EA are different. For a stand alone EA, all the processing (even evaluation time) is done on the host computer, so the speed of the host is a major factor in performance [13]. We will show that the speed of the host/central processor is not a major factor in intrinsic EHW but that measurement time is the major timing bottleneck. In addition, certain speedup methods, such as using parallel algorithms, that work with standalone EAs are difficult and expensive to implement in intrinsic EHW and thus are not currently used.

The third issue that makes intrinsic EHW unique is that there are often many possible solutions. Namely, with intrinsic EHW there are several possible configurations that give a correct or good result. For example, several different logic blocks on a FPGA can be used to form an AND operation. Likewise, on most FPGAs a single logic block can be configured in several ways to do an AND operation. Thus, the simple operation of ANDing together two inputs has several possible implementations in an FPGA and that does not include the spurious implementations that make use of capacitive coupling or other non-deterministic effects. Similarly in an FPAA, inputs can be routed to one of several possible input amplifiers without affecting the overall output. For example, to achieve an overall gain of, say, 12, the gains on two successive amplifiers can be set to any combination that multiply together to give 12:  $x_3$  then  $x_4$ ,  $x_6$  then

x2, x2.5 then x4.8. So instead of one unique optimum, the search space in intrinsic EHW often has several solutions that all have equal validity.

The fourth issue is that the design of the EA is not independent of the reconfigurable platform. For example, an EA that works and derives a good circuit on an FPGA will have to be modified and changed to work on an FPAA or even another type of FPGA. Specifically, the representation of the genome within the EA and the fitness function may both have to be changed. The changes could involve modifying the number of bits in the programming string, setting certain routing bits so that damage is not done to the device, changing the population size or other parameters so that the EA converges within a suitable time frame, or changing the structure of the EA entirely. The hardware platform imposes constraints and limitations on the type of EA that can be implemented.

All of these four issues make the problems associated with intrinsic EHW unique as compared to other types of evolutionary problems. Thus, there needs to be a model that accurately predicts the behavior of intrinsic EHW as it scales to larger problems and identifies the key issues associated with intrinsic EHW. This paper proposes such a model.

## 2.0 Size of design space

The goal is to determine the number of generations and the size of the population for a given programming string length,  $L$ . We develop a model for intrinsic EHW and derive equations for the number of generations,  $ngen$ , and the population size,  $N$ , given  $L$ . These equations for  $ngen$  and  $N$  are useful in three ways: (a) they allow us to compare the results to actual successful EHW experiments and thus confirm the model; (b) they allow us to set *a priori*  $N$  and  $ngen$  for future EHW experiments; (c) they allow us to see the complexity of  $N*ngen$  as a function of  $L$  and thus see why EHW does not scale well to medium or large circuits.

When calculating the expected number of generations till convergence and the population size, one must determine what type of model to use for the building blocks (BB) in the programming string. (Building blocks are short, highly fit bit strings within the longer programming string.) Two possible models have been evaluated in the literature [14-17]. In the first possible model, all the building blocks have a uniform weight. An example is the OneMax problem where all the bits (or building blocks) have a weight of 1 and the goal is to maximize the number of 1s in the bit string. This first model has uniform scaling [17]. In the second possible model, the BBs are exponentially scaled in such a way that the most significant BB has a weight that exceeds the sum of all the lower order BBs combined [14, 15]. An example of this model is the BinInt problem, where each bit (or BB) has a weight of  $2^m$  where  $m$  is the position of the bit. When a GA operates on this type of problem, the more significant bits (or BBs) converge faster than the less significant ones. Overall, there is a sequential convergence of bits that resemble a row of dominos falling down one after the other. (Hence, this type of convergence has been called domino convergence.)

We contend that the BBs in intrinsic EHW follow an exponential scaling such as the BinInt problem. There are two reasons for this contention. First, the nature of hardware-in-the-loop

evolution requires that certain parts of the problem be solved before other parts of the problem become relevant. A clear case is the issue of routing bits. Until the signal is routed to a particular amplifier, gate, switch or other device, it does not matter what the gain or function of that device is set to. Thus, the routing bits must be set and then the gains/device functions can be set to fine tune the fitness function. Thus, not all parts of the programming string are equally important or equally relevant. Some portions of the programming string (e.g., the routing bits) are responsible for large changes (and thus high variance) in the fitness function while others only affect the individual's fitness function by a small amount. In a similar vein, the gain bits for an amplifier in an FPAA tend to be binary weighted so even the bits within a BB are exponentially scaled. The second reason that we contend that the BBs are exponentially scaled is that it leads to good agreement with the experimental results from the literature. This agreement will be made clearer in section 2.3.

## 2.1 Number of generations

In this section we apply the model of Lobo et al. [14] to intrinsic evolvable hardware. The model makes the following assumptions:

- The programming string (chromosome) can be broken down into "subfunctions" or building blocks that are each  $k$  bits long.
- There is perfect BB mixing so a BB is not disrupted during crossover.
- The fitness of each BB corresponds to a needle in a haystack function where the one solution (the good solution) has a max fitness and all the other solutions have the same fitness,  $f_{min}$ .

Lobo et al. calculate the selection pressure for each generation of the GA. At each generation, a certain number of BBs,  $\lambda$ , have converged. Since the BBs are geometrically scaled, the most salient BBs will converge first so we can assume that the most salient  $\lambda$  BBs have converged and the remaining  $(\frac{L}{k} - \lambda)$  are still in their initial random state. It takes  $tgen$  generations for  $\lambda$  BBs to converge and they derive  $tgen$  as

$$tgen = \frac{-\ln 2}{\ln(1 - I \frac{1}{\sqrt{3(2^k - 1)}})} \lambda$$

In this equation,  $k$  is the size of each building block and  $I$  is the selection intensity. For a constant selection intensity, then, the number of generations till convergence is linear function of the importance or salience of the BB in the programming string.

To determine the number of generations till all BBs have converged,  $ngen$ , we set  $\lambda$  equal to the number of BBs in the string ( $\lambda = L/k$ ), and let the BB size,  $k$ , be 4. If binary tournament selection is used, then  $I = 1/\sqrt{\pi}$ . The number of generations till convergence is thus

$$ngen = \frac{-\ln 2}{\ln(1 - \frac{1}{\sqrt{3\pi(2^4 - 1)}})} \frac{L}{4} = 1.97L = K_1 L = O(L)$$

The value of the constant  $K_1$  is not critical at this point. Lobo et al. state specifically: “the constant factor obtained with the domino convergence model shouldn’t be taken too literally because this kind of modeling is not completely exact. Nevertheless, the functional form is correct (and is confirmed by experiments) and says that the average number of generations until convergence grows linearly with respect to the number of building blocks.” [14] In section 2.4.2 we determine an overall constant for  $N * ngen$  based on comparisons with experimental results and see that 1.97 is too large. But the conclusion is clear: the number of generations till all BBs converge is an  $O(L)$  process.

## 2.2 Derivation of population size

Several researchers have investigated the optimal population size for genetic algorithms [16-20]. Our goal is to determine the population size given the length of the programming string. We break the discussion into two cases: one for which the building blocks in the programming string are linearly scaled and one for which the building blocks are geometrically scaled.

### 2.2.1 Linearly scaled building blocks

If the building blocks are linearly scaled, the the approach of Harik et al. [17] can be used to determine population size,  $N$ , in terms of length of the chromosome string  $L$ . Harik develops a model of GAs based on an analogy between genetic algorithms and one-dimensional random walks. The result is an equation that relates the size of the population with the desired quality of the solution, as well as the problem size and difficulty. The final equation for  $N$  is given by [21]

$$N = 2^{k-1} \ln(\alpha) \frac{\sigma_{bb} \sqrt{\pi(L/k - 1)}}{d}$$

where

$\alpha$  = probability of GA failure

$k$  = building block order (length)

$L$  = length of programming string

$d$  = signal-to-fitness difference

$\sigma_{bb}$  = average root mean square building block standard deviation

Despite the rather complicated form of the equation, it does quantify many of the intuitive notions about the difficulties of GAs. For example, problems with long building blocks (long  $k$ ) are more difficult to solve than problems with short building blocks. Problems with a high variability ( $\sigma_{bb}$  high) are hard because it is difficult to detect the signal coming from the good solutions when the interference from not-so-good solutions is high. Longer problems (larger  $L$ ) are more difficult because there are more sources of noise. However, the GA scales to the problem size; the equation shows that the required population size grows linearly with the size of the problem.

For our purposes, we assumed that the programming strings are binary (or binary alphabets), the building block size is 4, the probability of GA failure is .01, the fitness difference between the best and 2<sup>nd</sup> best building block is 1, and the root mean square fitness variance ( $\sigma_{bb}^2$ ) is 0.04.

This yields an equation for population size as

$$N = 13\sqrt{\frac{L}{4} - 1},$$

where L is the length of the programming string. This result shows that N scales as the square root of L. The constant (13 in this case) will vary based on the problem difficulty (in general it will be a larger number) but N is still  $O(L^{1/2})$ . Also, even though we assumed a relatively simple problem difficulty when choosing parameters in the equation for N, the value of N still grows as  $O(L^{1/2})$ .

### 2.2.2. Geometrically scaled building blocks

An alternate derivation of population size must be used if the building blocks are geometrically scaled. When a GA operates on this type of problem, the more significant building blocks (genes) converge faster than the less significant ones. Given a large enough population size, all of the BBs will converge correctly one after the other. But if the population size is not large enough, then the GA may have trouble converging all the way down to the least significant BB [14]. The problem occurs because even when there is no selection pressure on a particular BB (the higher order BBs have not yet converged) the frequencies of the different BB combinations (alleles) will fluctuate due to chance variations and may eventually be lost completely from the population. This effect is known as random genetic drift.

The effects of the genetic drift can be modeled based on the physical process of diffusion [14]. The result is a second-order partial differential equation that gives a distribution of the number of population members with a certain BB value. From this distribution, one can derive an “extinction time” or the average number of generations to lose a building block. Given that we want to correctly solve all of the BBs, then the population size, N, must be selected large enough so that all BBs converge before random drift leads to the loss of one or more BBs. From Lobo et al.,

$$N = \frac{(2^k - 1)\lambda^*}{-2k \ln(1 - I \frac{1}{\sqrt{3(2^k - 1)}})}$$

where  $\lambda^*$  is the number of BBs that are solved correctly. In our case, we want all of the BBs to converge so  $\lambda^* = L/k$ ; letting  $k = 4$  and  $I = 1/\sqrt{\pi}$  (for binary tournament selection) leads to

$$N = 5.33L = K_2 L = O(L).$$

As with the equation for *ngen*, the value of the constant  $K_2$  is not necessarily 5.33 but the overall conclusion is clear: the population size grows linearly with the length of the programming string.

## 2.3 Comparison with experimental results

We contend that the BBs in intrinsic EHW will be geometrically scaled and thus follow a domino convergence pattern. If so, then the number of evaluations,  $N * ngen$ , will be  $O(L) * O(L)$  or  $O(L^2)$ . (If the BBs are linearly scaled, then the number of evaluations will scale as only  $O(L)$  [21].) To confirm our contention, we look at several successful intrinsic EHW experiments and look at the length of the programming string, the population size, and the number of generations to reach convergence (see Table 1). Also included in the table is some background information about the reconfigurable hardware device that was used, the intended application and the primary researcher who performed the experiment(s).

Device type	Program- ming string length, L	Pop size, N	# of gens, <i>ngen</i>	$N * ngen$ (est)	Application	Primary Researcher/ Reference
FPAAs (ispPAC30)	22	40	6-10	320	Averaging circuit for 3 sensors	Hereford [5]
FPAAs (ispPAC10)	27	20	60	1200	Analog feedback control system	Greenwood [9]
Custom (FPTA2)	154	100	97	9700	Motor controller	Gwaltney [8]
Custom (FPTA2)	154 (est)	100	80 – 100	9000	Half wave rectifier for sine wave	Ferguson [22]/ Stoica [4]
Custom (FPTA2)	924 (12 cells)	500	150- 200	87,500	4-bit digital-to- analog converter	Zebulum [23]
FPGA (XC6216)	1800	50	5000	250,000	Tone discriminator	Thompson [24]
Custom (FPTA)	2344	50	7000- 10000	425,000	6-bit digital-to- analog converter	Langeheine [10]

Table 1: Chart of successful intrinsic evolvable hardware experiments. Columns show the device type, the length of the programming string, the population size, the number of generations till convergence and then the application and researcher. Note: FPTA2 = Field Programmable Transistor Array from Jet Propulsion Lab, FPTA = Field Programmable Transistor Array from Germany.

We compared our model that relates the number of generations and population size based on the programming string length to the data in Table 1. The model says that the total number of evaluations should grow as  $L^2$ . To test the model we performed a regression analysis between  $N * ngen$  and  $L^2$ . That is, we determined  $\beta_0$  and  $\beta_1$  via regression in the following equation:

$$\text{Number of evaluations} = N * ngen = \beta_1 L^2 + \beta_0$$

Regression yielded  $\beta_0 = 7024$  and  $\beta_1 = .0761$  with a coefficient of determination,  $R^2$ , of .9980, so there is a very good agreement of the experimental data to the  $L^2$  curve. Figure 1 shows the visual comparison of our model prediction (solid line) to the results from successful intrinsic EHW experiments. Again, there is good agreement between the model and the data. We thus conclude that the total number of evaluations is an  $O(L^2)$  process.

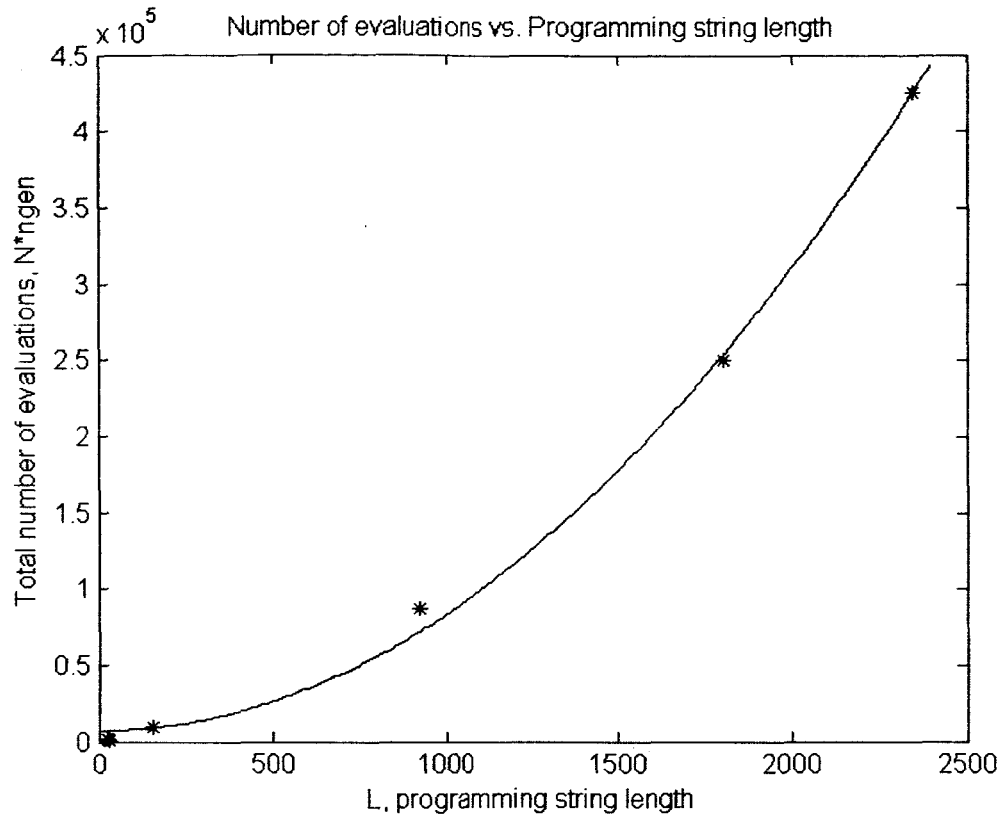


Figure 1: Plot of actual data points and  $O(L^2)$  curve fit for successful intrinsic EHW experiments. Asterisks show numbers from intrinsic EHW experiments and curve shows best fit  $O(L^2)$  curve.

There are many similarities and differences among the different experiments shown in Table 1 and Figure 1. A key similarity is that all of the researchers used a variation of the “simple” GA. That is, all members of the population were evaluated and then better ones were selected, single point crossover and mutation were applied and the cycle was repeated. Another similarity is that all of the researchers applied the crossover/mutation to the binary bit string directly, i.e., there were no special encodings between genotype and phenotype space. In addition, all the circuits that are derived during these EHW experiments are relatively simple circuits.

The differences among the different experiments are striking, however. There are five different reconfigurable platforms: two Field Programmable Analog Arrays (FPAA) (the ispPAC30 and the ispPAC10 from Lattice Semiconductor), the Field Programmable Transistor Array (FPTA2) (stand-alone board-level evolvable system – SABLES - from Jet Propulsion Lab), the Field Programmable Transistor Array (16x16 configurable array of CMOS transistor cells) and a Field

Programmable Gate Array (FPGA) (Xilinx XC6216). These are very different configurable devices with very different architectures. The FPAAs utilize internal analog amplifiers with analog inputs and analog outputs. The FPGA uses internal digital blocks with digital inputs and digital outputs. The FPTA2 is a fine-grain device that can be configured to handle analog inputs or digital inputs. And the programming string length varies by a factor of over 85 (almost 2 decades) in these successful experiments. Thus, our model has been shown to fit a wide range of platforms and string lengths.

We note that our model fits empirical data even though the equations for  $N$  and  $ngen$  are based on simplified implementations of statistical processes and genetic algorithms with crossover, but no mutation. Goldberg refers to these equations as facet wise models, because they isolate a single or small number of facets within the context of a larger problem [25]. These facet wise models give insight into the process of applying GAs to problem solving, and the models are verified using simulations of the GA and problem to be solved [17, 25]. However, the problem of autonomous circuit design is considerably more difficult than the BinInt problem used by Goldberg and his colleagues to develop the model. The relationship among the building blocks within the programming string/chromosome may not be easily identified in physical hardware. This makes the precise application of such analytical models more difficult, but in a general sense they still apply. The results obtained here for the number of evaluations,  $N*ngen$ , are estimates and, in general, represent the minimum number of evaluations needed.

## 2.4 Design space implications

### 2.4.1 Limiting size of design space

Based on our model of the design space, we can estimate how long it takes to program a device using evolvable hardware techniques. The following chart lists several reconfigurable devices that have been used or are being considered for intrinsic evolvable hardware. The table lists the devices,  $L$  (the number of bits required to do the programming),  $2^L$  (the total number of possible combinations given the number of programming bits),  $N*ngen$  from our design space model (using  $\beta_1 = .0761$  and  $\beta_0 = 7000$  as derived from regression), and estimated time to program the device assuming 50 ms per evaluation of each population member. The evaluation time will vary considerably based on application and experimental setup with 0.5 seconds per evaluation being close to worst case (see Section 3) and 0.05 seconds per evaluation being a nominal value.

Device	$L$	$2^L$	$N*ngen$ $O(L^2)$	Time
Pac30 (complete config)	112	$5 \times 10^{33}$	7955	6.6 mins
FPTA2 (limited)	500	$3 \times 10^{150}$	26,025	21 mins
FPTA2 (complete)	5000	$>10^{300}$	1,909,500	26 hrs
Virtex XCV50 (limited)	9600	$>10^{300}$	7,020,000	4 days
Virtex XCV50 (complete)	1,569,000	$>10^{300}$	$1.87 \times 10^{11}$	295 yrs

Table 2: Growth of design space and estimated time to evolve a design for various reconfigurable platforms.

From table 2, we conclude that intrinsic EHW will be limited to small and medium-sized circuits for the foreseeable future. To derive a large, moderately-complex circuit, the programming time for the reconfigurable device quickly grows beyond a reasonable value. For example, to evolve a circuit that utilizes half of the resources of the Virtex XCV50 would take over 50 years! The evaluation time is just too large to evolve completely a large circuit with current technology. In section 3 we explore where the timing bottleneck occurs and show that even a large speedup in processing speed does not significantly reduce the total evaluation time.

#### 2.4.2 Other limitations inherent in intrinsic evolvable hardware

Long evaluation times are one reason that it is difficult to evolve large circuits intrinsically. In addition to the time involved, there is also the possibility that an appropriate circuit will not be discovered. Two phenomena can intercede to preclude a hardware evolution from discovering a good circuit: genetic drift and non-uniqueness.

As mentioned in section 2.2.2, genetic drift occurs when there is no selection pressure on some of the BBs. In this case, one or more BBs will fluctuate stochastically and can eventually reach an “absorbing” state from which it can not reach the optimum value [14]. This can be a real problem with intrinsic EHW. Researchers must set the population size large enough so that all the BBs converge. If the population size is too small, then the best or even a suitable circuit may not be found. From the empirical results in Table 1, we see that the population size can vary considerably and still lead to successful results. To avoid genetic drift, however, we recommend that the population size be set to a relatively large value.

The issue of genetic drift has a practical consideration in picking the size of the design space. A common problem that occurs when it is known that there is a known good solution within the search space. For example, an adequate solution is possible with one cell on a programmable device. Will the results improve if the search space is expanded to include two cells? By expanding the search space (in this example, doubling the search space) then the programming string  $L$  will also double. This will lead to four times the number of circuit evaluations and also greatly increase the chances that genetic drift will lead to a suboptimal solution. Therefore, a researcher should pick as small design space as possible. If one block is sufficient to solve the problem, then do not pick 2 blocks and hope a better circuit will be found.

The second phenomenon is that there may be multiple possible solutions to the same circuit: non-uniqueness. Non-uniqueness implies that many different hardware configurations can lead to the same output. For example, in an experiment that derived a circuit to average three sensors, Hereford and Pruitt report that “there are many different redundant configurations that the genetic algorithm can find” [5]. Likewise, Thompson talks about “being able to exploit the subtle interactions between adjacent components that are not connected directly” [24]. This non-uniqueness is especially a problem with large design spaces. The genetic algorithm may find a solution that leads to a large fitness value but the solution may utilize parasitic coupling within the hardware device and not take into account all of the inputs or have some other spurious result. This would limit the applicability and transferability of the evolved circuit. What is needed for successful evolution is not only a well-defined peak in the search space but also a path to get to that peak. Multiple possible solutions within the device can lead to the GA

converging to a circuit solution that has a lot of “junk” elements or one that is not portable [24]. One way of overcoming this problem is to pick a proper fitness function. In addition, one must select the training data so that all possibilities are covered.

### 3.0 Timing issues

We want to consider how to reduce the evaluation time required for intrinsic evolvable hardware. In general, an EA used in intrinsic evolvable hardware has the following pseudo-code flowchart:

```

Begin EA
  Initialize population
  While (not done) do
    Evaluate each member of population
    Selection
    Crossover
    Mutation
  End while
End EA

```

For intrinsic EHW all of the steps can be done on the processing computer except the evaluation of each member of the population. The evaluation of each member requires the following steps:

```

Download bit string to device ( $t_{\text{down}}$ )
Update device (configuration, gains, routings) ( $t_{\text{update}}$ )
Measure new output ( $t_{\text{measure}}$ )
Read output and transfer to processor ( $t_{\text{read}}$ )

```

We thus define the evaluation time as

$$t_{\text{evaluate}} = t_{\text{down}} + t_{\text{update}} + t_{\text{measure}} + t_{\text{read}}$$

The  $t$ 's denote the time for each step. (Note: some implementations may not require each step in which case  $t$  can be set to zero for that step.) These steps must be done for each member of the population. If the population size is denoted  $N$ , then the calculation time for one loop of the EA is

$$t_{\text{cycle}} = N * (t_{\text{evaluate}}) + t_{\text{processor}}$$

where  $t_{\text{processor}}$  is the time for the processor to do selection, crossover, and mutation. The total calculation time is then

$$t_{\text{total}} = \text{ngen} * t_{\text{cycle}}$$

where  $\text{ngen}$  is the number of generations in the EA.

Consider the impact on the calculation time due to increased processor speeds. The metric we are trying to minimize is execution time, denoted above by  $t_{\text{total}}$  or, equivalently,  $t_{\text{cycle}}$ . The issue is that processor speed ( $t_{\text{processor}}$ ) is only one part of the equation, so improvements in processor

speed only affect one part of the whole process. By analogy with computer design, we can apply Amdahl's Law [26] which says

$$\text{execution time after improvement} = (\text{execution time affected by improvement/amount of improvement}) + \text{execution time unaffected}$$

For example, assume that processor speed increases by a factor  $n$  speedup. Then Amdahl's Law says that

$$t_{\text{cycle}}(\text{after speedup}) = \frac{t_{\text{process}}}{n \text{ speedup}} + n * (t_{\text{down}} + t_{\text{update}} + t_{\text{measure}} + t_{\text{read}})$$

In other words,  $t_{\text{cycle}}$  is not reduced by  $n$  speedup; rather, only  $t_{\text{process}}$  is reduced by  $n$  speedup. Since  $t_{\text{process}}$  is only a fraction of the  $t_{\text{cycle}}$ , increasing the processor speed has only minimal affect on the overall time.

To illustrate the (lack of) effect of increasing computer speed, we look at some sample numbers for two different intrinsic EHW setups. One setup utilizes a Field Programmable Analog Array (FPAA) from Lattice Semiconductor (ispPAC30) controlled by a PC. A digital multimeter (DMM) is used to read the output and a GPIB (IEEE 488) bus is used to transfer the data to the PC. This is an EHW setup using relatively slow devices. The second setup is the stand-alone board-level evolvable system (SABLES) developed by researchers at Jet Propulsion Laboratory [4, 22]. The SABLES system is a fast intrinsic EHW setup.

Table 3 gives sample values for the various time delays for the FPAA setup. Calculating  $t_{\text{cycle}}$  from these values (assuming a population size of 50) yields

$$t_{\text{cycle}} = 50 * (35.02) + .6 = 1751.6 \text{ msec} = 1.75 \text{ sec.}$$

Time	Value	Comment
$t_{\text{down}}$	12.6 usec	Time to download 112 bits from PC to pac30 over JTAG cable
$t_{\text{update}}$	9 usec	Time to erase and reprogram EECMOS cells and settling time of device
$t_{\text{measure}}$	10 msec	Integration time for DMM for 4.5 digit accuracy
$t_{\text{read}}$	25 msec	Transfer time for data from DMM to PC over GPIB bus
$t_{\text{process}}$	.6 msec	From timing measurements on Pentium III computer

Table 3: Sample values for the various time delays for the FPAA setup.

From Table 3 and the calculation of  $t_{\text{cycle}}$ , it is clear that the processor time (the time to do selection, crossover, etc) is a very small fraction of the total cycle time. Even if we used a new and fast computer that was, say 5 times faster ( $n$  speedup = 5) than the Pentium III, Amdahl's law says that the new execution time will be

$$t_{\text{cycle}}(\text{fast computer}) = 0.6/5 + 1751 = 1751.12 \text{ msec.}$$

Therefore, the speedup in processor speed will have an insignificant impact on total time in intrinsic evolvable hardware experiments. Clearly, much more can be gained in this example by decreasing the measurement and read times.

Now consider the cycle time for the fast (SABLES) EHW setup. From Stoica et al. [4], the stimulus/response time is 1.13 msec per population member. The GA processing is done using an on-board Digital Signal Processor (DSP) and Stoica et al. state that it takes 6 msec to generate a new population. Thus,

$$\begin{aligned} t_{\text{cycle}} &= 50 * (1.13 \text{ msec}) + t_{\text{processor}} \\ &= 62.5 \text{ msec.} \end{aligned}$$

It is clear that the SABLES setup provides an enormous speed advantage over the (slow) FPAA setup – it is approximately thirty times faster. But again Amdahl's Law shows that speeding up (i.e., reducing) the processor time will have only a minor impact on the overall execution time. Again assuming a speedup of 5 in the processor yields

$$t_{\text{cycle}}(\text{fast processor}) = 56.5 + 6/5 = 57.5 \text{ msec}$$

or only about an 8% time reduction for a large improvement in processor speed.

In general, using fast devices or architectures (such as SABLES [4], FPTA [10], or COMBO6 [11]) is a good approach for intrinsic EHW. However, one variable that can not always be arbitrarily reduced is the measurement time,  $t_{\text{measure}}$ . Its lower limit may be set by the settling/integration time of the measurement device (as in the FPAA example) or the lower limit may be set by the nature of the fitness function. For example, in a motor controller experiment performed in [8] several cycles of a low frequency sine wave must be monitored to calculate the fitness. The fitness function requires one cycle of a 2 Hz sine wave per individual or roughly 0.5 seconds per evaluation.

We have shown that improvements in processor speed will not significantly impact the total time in intrinsic EHW experiments. Processor speed will thus have only minimal impact on improving scalability in intrinsic EHW experiments. This is in contrast to Genetic Programming and other extrinsic EHW experiments where increases in processor speed have led to development of more complicated circuits [13].

## 4.0 System level design

The previous sections provide quantitative results for intrinsic hardware evolution. One result is the derivation of equations for  $N$  and  $n_{\text{gen}}$  given the length of the programming string,  $L$ . The second result is an analysis of the timing components for intrinsic EHW. These results are qualitatively intuitive based on experience and empirical data, but have not been quantified in published literature for specific devices used in intrinsic hardware evolution. In this section we discuss the implication of these results for practical implementation and give observations about the current state of practice for intrinsic EHW. We also discuss the idea of a "system design", where considerations for the interaction between the evolutionary algorithm (EA) and the reconfigurable platform contribute to the design of the overall evolutionary process.

From published literature we observe that when researchers use intrinsic hardware evolution, they are primarily concerned with hardware platform development. However, when researchers

are concerned with design of the evolutionary process, or with evolving a static design, such as an antenna, extrinsic hardware evolution is used. We define the evolutionary process to include an EA, representation of hardware in chromosomes, population sizing/initialization and fitness evaluation. Work involving the use of a physical platform such as the FPTA2, FPGAs, programmable analog arrays or multiplexed components routinely employ standard GAs, and in some cases standard Evolutionary Strategies (ES) [22, 27 - 31]. Researchers developing more complex approaches to the evolutionary process frequently use simulated hardware components. [32 - 35]. One reason for this may be that simulation lends itself more easily to implementation of more complex algorithms than a hardware platform with its resource and timing constraints. Further, some researchers are interested in rapid execution and implementation of the evolutionary process in hardware [36, 37]. This necessarily limits the complexity of the evolutionary process. The evolutionary process and the platform are intimately tied together and impose limits on each other. The platform imposes the real limitation of its performance capabilities and its level of reconfiguration. The evolutionary process imposes limitations due to design of the fitness function, representation of hardware in chromosomes and selection of operators.

As a simple example of the relationship between process and platform, consider an FPGA and an FPAA. The FPGA is designed specifically to implement digital circuits, while the FPAA is designed specifically to implement analog circuits. Assuming the same EA is used as the core of the evolutionary process for both devices, there will be different requirements for the design of the fitness function and the stimulus signals used for digital versus analog design. Further, if FPGA devices from two manufacturers are considered, there are in many cases significant differences in the configurable blocks within the device. One device may use a mixture of look-up tables to implement gates as well as fixed logic gates (Xilinx Virtex, Altera Stratix), while another may use routing only with fixed logic gates (Actel ProASIC). Even within similar configurable block approaches, the implementation of look-up table architectures varies with manufacturer. Chromosome representation will necessarily be affected by the architectural design of the reconfigurable device, and the representation may require the use of additional operators, or modifications to crossover and mutation operators. The evolutionary process must accommodate device performance capabilities in terms of parameters like bandwidth, electrical signal characteristics and possibility for damage. The reconfigurable device and the evolutionary process are a system that should be considered together in the design of an EHW platform. A system level approach to design that considers the characteristics of the device and the interaction of the EA with the device will contribute significantly to success in the evolution of complex systems.

It is pointed out in the literature that the standard GA in practice does not display the robustness described by Holland [25]. Thornton goes so far as to say that the schema theorem assumptions are hard to meet and the building block assumption violates the schema theorem such that the standard GA is only successful in cases, where it is guaranteed not to work effectively [38]. While Goldberg does not agree that the assumptions of the schema theorem cannot be easily met, he does agree the crossover operator in a standard GA employing selection and crossover often needs modification to work effectively. Further it is noted that choice of operators affects the success of the evolutionary process in discovering the "best" solution in a given application [17, 39, 40]. An overview of the variety of algorithms considered as being Evolutionary

Computation and the various forms of crossover, mutation and selection operators are given in [41]. Goldberg notes there is much "fiddling" with codings and modifications to operators that is reported by researchers who are seeking good results for a particular application [25]. The published literature acknowledges that development of the evolutionary process and the platform (simulated or real), along with the intended application of the evolved solution, are intimately tied.

Goldberg and his colleagues have concerned themselves with developing design guidelines for applying GAs to complex practical applications. Their work has demonstrated that GAs using fixed crossover operators require exponentially increasing populations to get good solutions to complex problems. A primary result of their research is that a selectorecombinative GA process using tractable population sizes must include the identification of linkage between bits in a chromosome before attempting to converge to a good solution. And, the GA should provide a method of emphasizing the linkage. An example of their efforts, the fast messy GA, addresses the issues of establishing linkage in a chromosome to identify building blocks and the need for rapid convergence in complex problems, while using a reasonable population size. The fast messy GA is an example of a so-called "competent GA" [25].

We note that many approaches for evolving complex circuitry involve using previously evolved circuits or known circuit configurations. In order to evolve a 4-bit digital-to-analog converter (DAC) on the JPL FPTA2, researchers used evolved circuits in a hierarchical approach along with human designed op-amp configurations. First a two-bit DAC was evolved and used as a building block for a 3-bit DAC. This 3-bit DAC is used as a building block for a 4-bit DAC. Op-amp configurations are used for amplification and buffering [23]. Previously, a 4-bit DAC was evolved hierarchically in simulation on a Sun SPARC 2 workstation. This experiment demonstrated building block encapsulation as a method of reducing the total number of evaluations needed to evolve the 4-bit DAC [42]. Torreson applied similar principles to evolve systems for character recognition and prosthetic hand control [43]. Koza, et. al., use the concept of sub-circuit re-use in the evolution of analog filters using Genetic Programming (GP) [32]. A similar notion is explored in the evolution of a 3-bit multiplier by Vassilev and Miller using Cartesian GP with a  $(1 + \lambda)$  Evolutionary Strategy (ES) [44]. This approach could be applied to intrinsic EHW. Time for evolution is reduced by using already evolved components, rather than forcing the evolutionary algorithm to re-evolve basic circuits. Humans routinely employ such a strategy in the design of electronic circuits. While such approaches make sense from a practical standpoint, they are viewed as limiting the design space. This view of design space defines such space as the genotype-to-phenotype representation imposed by the hardware platform or simulation environment. Reconfigurable electronics do impose restrictions on the design space due to the construction of routing resources and functional components, but perhaps this definition of design space is too narrow, being only one component of a system for hardware evolution.

Expanding on the idea of component re-use, a large design can, in some cases, be partitioned into sub-designs. Each design partition could then have an EA which acts only on that section. This is especially applicable to digital circuitry, where the design partitions can be easily evaluated separately. The partitions could include previously evolved components in combination with sections to be evolved to meet new requirements. In this way a parallel effort by several EAs

would reduce overall time for evolution. Creating an automated method for determining when to use previously evolved components in lieu of evolving new components can provide a means of reducing evolution time without limiting the design space of the reconfigurable device. This system design approach will require careful design of evaluation cases and fitness functions.

Finally, the EA used in intrinsic hardware evolution will need to be prepared to “fire on all cylinders” and achieve the rapid convergence that will be required in deployed systems. The efforts of Goldberg and other EA theorists should be leveraged in the creation of a platform to rapidly, and consistently, evolve good solutions in reconfigurable hardware. These EAs will be more complex than those typically used in intrinsic EHW. But, there is an abundance of fast processors that can be used to provide an embedded solution that will not allow the increased processing needed for such complex EAs to have a significant effect on the total time for the evolutionary process. Evaluation time will still be the most significant parameter, and can be directly addressed by reducing the total number of evaluations needed via the use of “competent EAs” to improve the evolutionary process.

The analytical development in section 2 does not address the system design space for intrinsic hardware evolution. It would be difficult to do so in an equation. But, it does give us insight into a portion of the problem we are addressing, and reinforces our feeling that the intrinsic EHW community is on the right track. Equations that estimate time to convergence can be useful in a strategy for automatically determining convergence. In order to be successful in the evolution of complex circuits, a balance between EA theory and implementation of intrinsic hardware implementation will have to be struck. Ultimately, a compromise will be made between the restriction of design space and the capability for innovation in design.

## 5.0 Conclusions

Scalability (the ability to derive larger and more complex circuits) is a big issue in intrinsic EHW. In this paper we derived equations that determine how the design space,  $N^{ngen}$ , grows as the programming string gets longer. We have shown that the design space grows as  $L^2$ . Thus, to derive even moderately-complex circuits, the programming time for the reconfigurable device quickly grows beyond a reasonable value. And to evolve a circuit that utilizes even half of the logic blocks on an FPGA would take dozens of years!

In addition to programming time, long programming strings lead to other problems that can preclude the EA from finding a good circuit. First is the problem of genetic drift. That is, some of the lower order BBs drift randomly (no selection pressure) till they converge to a suboptimum value. When the more salient BBs finally converge, these low order BB(s) can not then get out of their present state except through the operation of mutation. The second problem is that there are multiple possible solutions to the same circuit. Multiple possible solutions within the device can lead to the GA converging to a circuit solution that has a lot of “junk” elements or one that is not portable [24]. The lesson here is to pick the smallest possible design space to implement a given circuit.

For other types of Evolutionary Algorithms such as Genetic Programming, more complex circuits have been derived as processor speeds have increased [13]. For intrinsic EHW,

however, processor speed is not the long pole in the timing equation. Instead, evaluation time (the time to evaluate each member of the population) is the biggest factor. Increasing processor speeds by even 5 times will not have a significant impact (i.e., more than 10%) on the overall time to implement EHW. Thus, the SABLES approach, where they strive to reduce the evaluation time, is a good approach to intrinsic EHW.

Several researchers have looked at the problem of scalability for intrinsic EHW. From a review of the published literature, we observe that researchers using intrinsic EHW are primarily concerned with hardware platform development but researchers concerned with the evolutionary process use extrinsic hardware simulation/evolution. The evolutionary process and the type of reconfigurable hardware are intimately tied together and impose limits on each other. These two components of hardware evolution are a system and need to be considered together in the design of an intrinsic EHW platform.

We also observe that the equations for  $N^*ngen$  are estimates and represent the minimum evaluations needed. Goldberg and colleagues have pointed out that a selectorecombinative GA process must include the identification of linkage between bits the string (chromosome) before attempting to converge to a good solution. This is a significant consideration in the design of evolutionary algorithms to support evolvable hardware.

A third observation is that many approaches for evolving complex circuits involve using previously evolved circuits or known circuit configurations. The time for evolution is reduced by using already evolved components. While such approaches make sense from a time standpoint, they can be viewed as limiting the design space, especially when they involve fixing portions of a reconfigurable device.

### **Acknowledgements**

Dr. Hereford would like to thank the Kentucky NSF EPSCoR Research Enhancement Grant program and Murray State University for support. Mr. Gwaltney is supported by a NASA Marshall Space Flight Center Director's Discretionary Fund task.

### **References**

1. X. Yao and T. Higuchi, "Promises and challenges of evolvable hardware", IEEE Transactions on Systems, Man, and Cybernetics – Part C: Applications and Reviews, Vol. 29, no. 1, pp. 87 – 97, February 1999.
2. A. Thompson, I. Harvey, P. Husbands, "The natural way to evolve hardware", Proceedings IEEE International Symposium on Circuits and Systems, 1996, pp. 37 – 40.
3. T. Higuchi, M. Iwata, D. Keymeulen, H. Sakanashi, M. Murakawa, I. Kajitani, E. Takahashi, K. Toda, M. Salami, N. Kahihara, N. Otsu, "Real-World applications of analog and digital evolvable hardware", IEEE Transactions on Evolutionary Computation, Vol. 3, No. 3, pp. 220 – 235, September 1999.

4. A. Stoica, R. Zebulum, M. I. Ferguson, D. Keymeulen, V. Duong, X. Guo, "Evolving circuits in seconds: Experiments with a stand-alone board-level evolvable system", 2002 NASA/DoD Conf. on Evolvable Hardware, July 2002, pp. 67-74.
5. J. Hereford, C. Pruitt, "Robust sensor systems using evolvable hardware", 2004 NASA/DoD Conference on Evolvable Hardware, Seattle, WA, Zebulum et al. (ed.), 2004, pp. 161 - 168.
6. R. O. Canham, A. Tyrrell, "Evolved fault tolerance in evolvable hardware", IEEE Congress on Evolutionary Computation 2002, Honolulu, HI, 2002, pp. 1267 – 1272.
7. S. J. Flockton, K. Sheehan, "Evolvable hardware systems using programmable analogue devices", *IEE Colloquium Digest*, 1998, pp. 511 – 516.
8. D. Gwaltney, M. Ferguson, "Intrinsic Hardware Evolution for the Design and Reconfiguration of Analog Speed Controllers for a DC Motor", 2003 NASA/DoD Conf. on Evolvable Hardware, Chicago, IL, July 2003, pp. 81-90.
9. G. Greenwood, D. Hunter, E. Ramsden, "Fault recovery in linear systems via intrinsic evolution", 2004 NASA/DoD Conference on Evolvable Hardware, Seattle, WA, Zebulum et al. (ed.), 2004, pp. 115 – 122.
10. J. Langeheine, K. Meier, J. Schemmel, M. Trefzer, "Intrinsic evolution of digital-to-analog converters using a CMOS FPTA chip", in 2004 NASA/DoD Conference on Evolvable Hardware, Seattle, WA, Zebulum et al. (ed.), 2004, pp. 18 – 25.
11. L. Sekanina, S. Friedl, "On routine implementation of virtual evolvable devices using COMBO6", 2004 NASA/DoD Conference on Evolvable Hardware, Seattle, WA, Zebulum et al. (ed.), 2004, pp. 63 - 70.
12. J. He, X. Yao, "Drift analysis and average time complexity of evolutionary algorithms", *Artificial Intelligence*, vol. 127, pp. 57 – 85, 2001.
13. J. Koza, M. Keane, M. Streeter, W. Mydlowec, J. Yu, G. Lanza, Genetic Programming IV: Routine Human-Competitive Machine Intelligence, Kluwer, 2003.
14. F. Lobo, D. Goldberg, M. Pelikan, "Time complexity of genetic algorithms on exponentially scaled problems", *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, New York, NY, 2002, pp. 151- 158.
15. D. Thierens, D. Goldberg, A. Pereira, "Domino convergence, drift, and the temporal-salience structure of problems", *Proceedings of the 1998 IEEE International Conference on Evolutionary Computation*, 1998, pp. 535 – 540.
16. D. E. Goldberg, K. Deb, J. H. Clark, "Genetic Algorithms, noise, and the sizing of populations", *Complex Systems*, vol. 6, pp 333 – 362, 1992.

17. G. Harik, E. Cantu-Paz, D. E. Goldberg, B. L. Miller, "The gambler's ruin problem, genetic algorithms, and the sizing of populations", *Evolutionary Computation*, vol. 7, number 3, pp. 231 – 253, 1999.
18. Grefenstette, J. J., "Optimization of control parameters for genetic algorithms", *IEEE-SMC, SMC-16*, pp. 122-128, 1986.
19. J. D. Schaffer, R. A. Caruana, L. J. Eshelman and R. Das, "A study of control parameters affecting online performance of genetic algorithms for function optimization", *Proc of 3<sup>rd</sup> International Conf. on Genetic Algorithms*, 1989.
20. J. T. Alander, "Population size, building blocks, fitness landscape and genetic algorithm efficiency in combinatorial optimization: an empirical study" in *Practical Handbook of Genetic Algorithms*, vol III, ed. by Lance Chambers, CRC Press, pp. 459 – 485, 1999.
21. J. Hereford, D. Gwaltney, "Design space issues for intrinsic evolvable hardware", 2004 NASA/DoD Conference on Evolvable Hardware, Seattle, WA, Zebulum et al. (ed.), 2004, pp. 231 - 235.
22. M. Ferguson, R. Zebulum, D. Keymeulen, A. Stoica, "An Evolvable Hardware Platform Based on DSP and FPTA", *Late Breaking Papers at the Genetic and Evolutionary Computation Conf. (GECCO-2002)*, July 2002, pp. 145-152.
23. R. Zebulum, D. Keymeulen, V. Duong, X. Guo, M. Ferguson, A. Stoica, "Experimental results in evolutionary fault-recovery for field programmable analog devices", 2003 NASA/DoD Conference on Evolvable Hardware, Chicago, IL, Lohn et al. (ed.), 2003, pp. 182-186.
24. A. Thompson, P. Layzell, R. Zebulum, "Explorations in design space: Unconventional electronics design through artificial evolution", *IEEE Transactions on Evolutionary Computation*, Vol. 3, no. 3, pp. 167 – 196, September 1999.
25. David E. Goldberg, The Design of Innovation Lessons From and For Competent Genetic Algorithms, Kluwer Academic Publishers, Boston, MA USA, 2002.
26. David Patterson, John Hennessy, Computer Organization and Design: The Hardware/Software Interface, Morgan Kaufmann, San Francisco, 1998.
27. J. Lohn, G. Larchev, R. DeMara, "Evolutionary Fault Recovery in a Virtex FPGA Using a Representation that Incorporates Routing", *International Parallel and Distributed Processing Symposium (IPDPS'03)*, France, April 22 - 26, 2003.
28. J. F. Amaral, J. L. Amaral, C. Santini, R. Tanscheit, M. Vellasco, M. Pacheco, A. Mesquita, "Evolvable Building Blocks for Analog Fuzzy Logic Controllers", 2003 NASA/DoD Conf. on Evolvable Hardware, Chicago, IL, July 2003, pp. 101-107.

29. G. Greenwood, E. Ramsden, S. Ahmed, "An Empirical Comparison of Evolutionary Algorithms for Evolvable Hardware with Minimum Time-to-Reconfiguration Requirements," 2003 NASA/DoD Conf. on Evolvable Hardware, Chicago, IL, July 2003, pp.59 – 65.
30. P. Layzell, "Reducing Hardware Evolution's Dependency on FPGAs," Proc of MicroNeuro '99, 7th International Conf. on Microelectronics for Neural, Fuzzy and Bio-inspired Systems. IEEE Computer Society, CA. April 1999, pp171-178.
31. J. Langeheine, K. Meier, J. Schemmel, Intrinsic "Evolution of Quasi DC solutions for Transistor Level Analog Electronic Circuits Using a CMOS FTPA Chip," 2002 NASA/DoD Conf. on Evolvable Hardware, July 2002, pp. 75-84.
32. J. Koza, M. Keane, and M. Streeter, "The importance of Reuse and Development in Evolvable Hardware", 2003 NASA/DoD Conf. on Evolvable Hardware, July 2003, pp. 33-42.
33. J. Lohn, G. Haith, S. Colombana, D. and Stassinopoulos, "Towards Evolving Electronic Circuits for Autonomous Space Applications," Proc of the 2000 IEEE Aerospace Conf., Big Sky, MT, March 2000, pp 476-486, Vol. 5.
34. J. Botelho, L. Sa, P. Vieira, A. Mesquita, "An Experiment on Nonlinear Synthesis Using Evolutionary Techniques Based only on CMOS Transistors," 2003 NASA/DoD Conf. on Evolvable Hardware, Chicago, IL, July 2003, pp. 50-58.
35. S. Louis, "Learning for Evolutionary Design", 2003 NASA/DoD Conf. on Evolvable Hardware, Chicago, IL, July 2003, pp. 17 – 20.
36. T. Higuchi, M. Iwata, E. Takahashi, Y. Kasai, H. Sakanashi, M. Murakawa, I. Kajitani, "Development of Evolvable Hardware at Electrotechnical Laboratory," IEEE International Conf. on Industrial Electronics, Control and Instrumentation, 2000.
37. J. Gallagher, "The Once and Future Analog Alternative: Evolvable Hardware and Analog Computation," 2003 NASA/DoD Conf. on Evolvable Hardware, Chicago, IL, July 2003, pp. 43-49.
38. Chris Thornton, "The building block fallacy", Complexity International, 4, 1997.
39. David Fogel, Evolutionary Computation Principles and Practice for Signal Processing. SPIE Press, Bellingham, Washington USA. 2000.
40. J. Grefenstette, "Evolvability in Dynamic Fitness Landscapes: A Genetic Algorithm Approach," Proc. 1999 Congress on Evolutionary Computation (CEC 99), Washington, DC., pp. 2031-2038.
41. T. Back, H.-P. Schwefel, "Evolutionary Computation: An Overview," Proc of the Third IEEE Conf. on Evolutionary Computation 1996, pp. 20-29, IEEE Press, Piscataway NJ, 1996.

42. A. Stoica, R. Zebulum, D. Keymeulen, M. Ferguson, X. and Guo, "Scalability Issues in Evolutionary Synthesis of Electronic Circuits: Lessons Learned and Challenges Ahead," AAAI Spring Symposium on Computational Synthesis, Stanford University, CA, March 24-26, 2003.
43. J. Torreson, "A scalable approach to evolvable hardware", Genetic Programming and Evolvable Machines, vol. 3, pp. 259 – 282, 2002.
44. V. Vassilev and J. Miller, "Scalability Problems of Digital Circuit Evolution," Proc. of the 2nd NASA/DOD Workshop on Evolvable Hardware, Los Alamitos, California, U.S.A., 2000.